APPLICATION

FOR

UNITED STATES PATENT

Entitled

A Method And Apparatus Utilizing Non-Uniformly Distributed DRAM Configurations And To Detect In-range Memory Address Matches

Inventors:

Chen-Chi Kuo
Sridhar Lakshmanamurthy
Rohit Natarajan
Kin-Yip Liu
Prashant Chandra
James Guilford

David W. Rouille
Daly, Crowley & Mofford, LLP
275 Turnpike Street, Suite 101
Canton, Massachusetts 02021-2310
Telephone (781) 401-9988 x25
Facsimile (781) 401-9966

Intel Corporation
Intel Case No.: P17937
Attorney Docket No.: INTEL-010PUS

TITLE OF THE INVENTION

A Method And Apparatus Utilizing Non-Uniformly Distributed DRAM Configurations And To Detect In-range Memory Address Matches

CROSS REFERENCE TO RELATED APPLICATIONS

Not Applicable

STATEMENT REGARDING FEDERALLY SPONSORED RESEARCH

Not Applicable

FIELD OF THE INVENTION

The present disclosure relates generally to memory channel organization and allocation for use by a network processor and more specifically to memory channel organization and utilization in a non-uniformly distributed configuration as well as to a method for detecting in-range memory address matches.

BACKGROUND OF THE INVENTION

The Internet, as well as other networks, has become more complicated with additional bandwidth requirements, a larger number users, and increasingly elaborate uses. In order to handle these increased demands, new protocols and network data types have been developed. Network Processors (NPs), which are well known to those of ordinary skill in the art, are used to perform various tasks such as processing network packets, network data streams, and network objects to accomplish specific tasks.

The functions that the NP performs can be generally categorized into physical-layer functions, switching and fabric-control functions, packet-

processing functions, and system control functions. In some instances, the packet-processing functions can be further subdivided into network-layer packet processing and higher-layer packet processing.

The physical-layer functions handle the actual signaling over the network media connections, such as an Ethernet port, an optical fiber connection, or a coaxial T3 connection. The NP converts the data packets into signals that are transmitted over the physical media. These often work according to a media access control (MAC) and physical layer protocols such as Ethernet, Synchronous Optical Network (SONET), Time Division Multiple Access (TDMA), Code Division Multiple Access (CDMA), and the like.

The switching and fabric-control functions performed by the NP direct data from the ingress port of the NP to an appropriate egress port of the NP. Further functions include performing operations, such as queuing the data in appropriate order or priority at these ports.

The packet-processing functions performed by the NP handle the processing of various network protocols. Thus, a packet containing instructions on allocating a stream for continuous guaranteed delivery is handled at this level.

System-control or host-processing functions performed by the NP include the management of other components of the hardware unit, such as power management, device control, console port management, and the like.

NP processing typically includes other functions as well. A typical router application involves receiving packets, performing route table look-ups, packet

classification, packet metering, congestion avoidance, packet transmit scheduling and packet transmittal. NPs should provide sufficient processing power in order to run network applications efficiently and cost-effectively.

BRIEF DESCRIPTION OF THE DRAWINGS

The foregoing features of the present disclosure may be more fully understood from the following description of the drawings in which:

Figure 1 is a block diagram of a network processor system;

Figure 2 is a flow chart of the method of determining a memory channel and physical block location within the channel for memory addressing;

Figure 3A is a first part of a flow diagram of the method of determining channel ownership and block allocation within a channel; and

Figure 3B is a second part of the flow chart of Figure 3A;

Figure 4 is a flow diagram of a method to perform mapping of a program address into a physical address within a memory channel;

Figure 5 is a diagram showing address matching within a predetermined range; and

Figure 6 is a flow diagram of a method to detect in-range memory address matches.

DETAILED DESCRIPTION OF THE INVENTION

Referring to Figure 1, a memory unit in a network processor 10 is responsible for controlling the off chip DRAM 30, 40 and 50 and provides a mechanism for other functional units in the network processor to access the DRAM. A network processor 10 may have multiple controllers each of which manages one of the DRAM channels.

Typically, DRAM channels associated with a network processor are used for packet data buffering. For a typical application utilizing an NP, cells/packets arrive at a specified line rate. At a line rate of 2.5 Gigabytes per second (Gbps) a 40 byte packet may arrive every 16 nanoseconds (ns). As line rates increase there is a concomitant increase in packet arrival rate. At a line rate of 10 Gbps packet arrival increases to every 4 ns, and at a line rate of 40 Gbps the packet arrival rate increases to 1 ns. The NP performs the layer 3 through layer 7 processing on these packets. The NP also transmits the processed packets in the desired sequence and at the desired rate. If the NP cannot keep up with the arrival rate, packets are dropped or otherwise lost.

Each of the controllers independently accesses its own DRAMs, and can operate concurrently with the other controllers (i.e. they are not operating as a single, wider memory). The memory space is guaranteed to be contiguous from a software perspective. Hardware interleaving (also known as striping) of addresses is done to provide balanced access to all populated channels with DRAM commands. Each channel is striped at 128-byte blocks, although it should be appreciated that other striping sizes may also be used.

Interleaving is a technique used to improve memory performance. Memory interleaving increases bandwidth by allowing simultaneous access to more than one chunk of memory. This improves performance because the processor can transfer more information to/from memory in the same amount of time, and helps alleviate the processor-memory bottleneck that can be a major limiting factor in overall performance.

Interleaving works by dividing the memory into multiple channels. When a read or write is begun to one channel, a read or write to other channels can be overlapped with the first one. In order to get the best performance from this type

of memory system, consecutive memory addresses are spread over the different channels.

While the exemplary embodiment shown and describes herein shows three memory channels, it is understood that any number of memory channels could be used. To fully utilize the address space the network processor supports a non-uniformly defined configuration wherein channel 0 is assigned twice the memory capacity of channel 1 and wherein channel 2 is assigned the same capacity as channel 1. Another reason the non-uniform configuration is used is due to the limited granularity of DRAM modules. For example, there is no DRAM module having a size of 1.33GB to uniformly populate three channels and fully utilize a 4GB address space. For example, a system of 4G-Byte memory is populated with 2G-Byte, 1G-Byte, and 1G-Byte capacity respectively in channel 0, 1, and, 2. Table 1 below shows the allocation of memory space within the three channels for different memory sizes

TABLE 1

Total DRAM capacity in 3 channels	**channel 0**	**channel 1**	**channel 2**
512MB	256MB	128MB	128MB
1GB	512MB	256MB	256MB
2GB	1GB	512MB	512MB
4GB	2GB	1GB	1GB

When all three channels are active and one of the channels is populated with twice the capacity as in the other two channels, the interleave scheme selecting the channel for each block is shown in Table 2 and in the code segment below.

TABLE 2 Bits for Channel Selection and Remapping Location for Different Memory Configurations

Total DRAM capacity in 3 channels	bit[a:b]	Address (X)
512MB	bit[28:27]	128MB
1GB	bit[29:28]	256MB
2GB	bit[30:29]	512MB
4GB	bit[31:30]	1GB

5

Address Re-arrangement for Non-uniform Configurations with Three Active Channels

Line 1: // bit[a:b] and Address(X) are configuration-specific, as shown in Table 2.
Line 2: if(address bit[a:b] == 11){
Line 3: the address is mapped to channel 0;
Line 4: the location in the channel 0 is Address(X) + (~(0x3<<b)&address)
Line 5: }
Line 6: else{
Line 7: use sum modulo-3 reduction shown in Figure 3 to decide channel;
Line 8: (address bits [31:7] are summed as modulo 3, and the remainder is the
Line 9: selected channel number. This ensures that adjacent blocks are mapped to
Line 10: different channels.)
Line 11: use Table 3 to decide address re-arrangement;
Line 12: }

Lines 1-5 of the code segment are used for program addresses that will map into a physical address within the second half (non-interleaved portion) of the first channel. For example a write to a program address of CF000000 will be

mapped to physical address 4F000000 within channel 0 for a system having 4 GB of memory wherein channel 0 has 2GB of memory while channels 1 and 2 have 1GB of main memory each. It is understood that addresses are in Hexadecimal unless otherwise stated.

In this scenario, bits [a:b] correspond to bits [31:30] and Address X is 40000000. Line 4 of the code segment states that the physical location in channel 0 that the program address maps to is: Address X + (~(0x3<<b)&address). The portion of the code that states "~ (0x3<<b)" translates to left shifting the bits 0x3a total of b times and negating the value. Since b is 30, the 0x3 is right shifted 30 times and negated, then the remainder of the address is appended to it, giving the value 5F000000. This is then added to Address X (40000000 in this example) yielding a physical address within channel 0 of 1F000000.

Lines 6-12 of the code segment are used to interleave program addresses into the appropriate channel of the lower ¾ of memory. Address bits 31:7 are summed as modulo 3 wherein the remainder is the appropriate channel number. The program address must then be mapped into a physical address within the selected channel. This is done in accordance with table 3 below.

TABLE 3

Address Rearrangement for 3 Way Interleave

When these bits of address are all "1"s.	Shift 31:7 right by this many bits	Add this amount to shifted 31:7 (based on amount of memory on the channel) Address within channel is {shifted 31:7+table_value), 6:0}				
		64MB	128MB	256MB	512MB	1GB
30:7	26	N/A	N/A	N/A	N/A	8388607
28:7	24	N/A	N/A	2097151	4194303	8388606
26:7	22	524287	1048575	2097150	4194300	8388600
24:7	20	524286	1048572	2097144	4194288	8388576
22:7	18	524280	1048560	2097120	4194240	8388480
20:7	16	524256	1048512	2097024	4194048	8388096
18:7	14	524160	1048320	2096640	4193280	8386560
16:7	12	523776	1047552	2095104	4190208	8380416
14:7	10	522240	1044480	2088960	4177920	8355840
12:7	8	516096	1032192	2064384	4128768	8257536
10:7	6	491520	983040	1966080	3932160	7864320
8:7	4	393216	786432	1572864	3145728	6291456
None	2	0	0	0	0	0

An example of performing the channel determination and block allocation within the channel for the non-uniformly allocation of memory between three channels will be discussed. While the example described below uses three memory channels with a total of 4 GB of memory distributed such that channel 0 contains 2GB of memory, channel 1contains 1 GB of memory and channel 2 contains 1 GB of memory, it is understood that the same concepts apply to other systems having different number of memory

channels, different amounts of memory or a different allocation of memory between the channels.

In this example the program address to be mapped is address 25B42F80. Since this example uses 128 byte blocks for channel striping, bits 6:0 of the program address comprise the byte offset and are not used as part of the channel determination. Bits 30:7 of the address 25B42F80are represented as shown:

010 0101 1011 0100 0010 111 11

The above address bits are grouped into pairs and a 2 bit to 2 bit recoding is performed in accordance with Table 4 below.

TABLE 4

2 bit group to be recoded	2 bit recode
00	00
01	01
10	10
11	00

Thus the above address bits 30:7 are transformed from:

01 00 10 11 01 10 10 00 01 01 11 11

to:

01 00 10 00 01 10 10 00 01 01 00 00

This set of binary digits is then grouped into groups of four:

0100 1000 0110 1000 0101 0000

At this stage of the process, a 4 bit–to-2 bit reduction is performed on the above data in accordance with Table 5 below:

TABLE 5

4 bit group to be reduced	2 bit reduction
0000	00
0001	01
0010	10
0100	01
0101	10
0110	00
1000	10
1001	00
1010	01

The resulting set of 12 digits after performing the 4 bit to 2 bit reduction is 01 10 00 10 10 00. This set of 12 digits is then separated into groups of four to get 0100 0010 1000. The 4 bit-to-2 bit reduction is performed again (using Table 5 again), to arrive at 00 10 10.

At this stage of the process, an interim channel number is assigned a binary value of "00". The interim channel number is pre-pended to the 6 bit address 001010 giving a value of 00001010. A 4 bit to 2 bit reduction is

performed to arrive at a 4 bit result which is 00 01. A 4 bit to 2 bit reduction is performed again, yielding a result of 01.

Since the result of the above operation was non-zero, the interim channel number is incremented (now has a binary value of "01"), and this interim channel number is pre-pended to the 6 bit address 001010, giving a value of 01001010. A 4 bit to 2 bit reduction is performed to arrive at a 4 bit result which is 0101. A 4 bit to 2 bit reduction is performed again, yielding a result of 10.

Since the result of the above operation was non-zero, the interim channel number is incremented again (now has a binary value of "10"), and this interim channel number is pre-pended to the 6 bit address 001010, giving a value of 10001010. A 4 bit to 2 bit reduction is performed to arrive at a 4 bit result which is 1001. A 4 bit to 2 bit reduction is performed again, yielding a result of 00. Since this result was "00", the interim channel number of "10", which is not incremented here, is the channel number that the program address will be mapped into, thus the program address will be mapped into channel 2. The above process performs a modulo 3 arithmetic operation to arrive at a channel number for the program address.

Once the channel number is determined, table 3 is used to map the program address to a physical address within the channel. While the example below describes mapping an address into one of three memory channels with a total of 4 GB of memory distributed such that channel 0 contains 2GB of memory, channel 1contains 1 GB of memory and channel 2 contains 1 GB of memory, it is understood that the same concepts apply to other systems having a

different number of memory channels, different amounts of memory or a different allocation of memory between the channels.

In this example bits 10:7 are all "1"s, therefore bits 31:7 are right shifted by 6 bits, yielding the number 0296D0BE. Next the appropriate value from table 3 (values in table 3 are decimal, and therefore need to be converted) is added to 0296D0BE. This value is 7864320. The addition of 7864320 to 0296D0BE yields 0A1D13DE. This value then has bits 6:0 appended to the end of it to yield the physical address within the channel, namely physical address 050E89EF00. Therefore, in accordance with one presently disclosed method for utilizing non-uniformly distributed memory in a NP, the program address of 25B42F80 is mapped into physical address 050E89EF0 in memory channel 2.

A flow chart of the presently disclosed method is depicted in Figures 2, 3A and 3B. The rectangular elements are herein denoted "processing blocks" and represent computer software instructions or groups of instructions. The diamond shaped elements, are herein denoted "decision blocks," represent computer software instructions, or groups of instructions which affect the execution of the computer software instructions represented by the processing blocks.

Alternatively, the processing and decision blocks represent processing performed by functionally equivalent circuits such as a digital signal processor circuit or an application specific integrated circuit (ASIC). The flow diagrams do not depict the syntax of any particular programming language. Rather, the flow diagrams illustrate the functional information one of ordinary skill in the art requires to fabricate circuits or to generate computer software to perform the processing required in accordance with the present method and apparatus. It

should be noted that many routine program elements, such as initialization of loops and variables and the use of temporary variables are not shown. It will be appreciated by those of ordinary skill in the art that unless otherwise indicated herein, the particular sequence of processing described is illustrative only and can be varied. Thus, unless otherwise stated the processing described below is unordered meaning that, when possible, the processing can be performed in any convenient or desirable order.

Referring now to Figure 2, a process for determining channel ownership and physical block location within the channel in a non-uniformly distributed DRAM configuration is shown.

The process 100 starts and processing block 110 is executed. In processing block 110 the two most significant bits (MSBs) of the address for the maximum address of the DRAM channels is determined. For example, if there was a total memory capacity of 4GB in the three memory channels, than the MSBs would be bits 31:30. In the scenario where there is a total memory capacity of 1GB in the three memory channels than the MSBs would be address bits 29:28

Decision block 120 is executed next wherein a determination is made whether the MSBs of the program address which is going to be mapped into a memory channel are both "1"s. When the MSBs are both "1"s, then processing continues with processing block 130. When the MSBs are not both "1"s, then processing continues with processing block 140.

When the MSBs of the program address are both "1"s (indicating the address will reside in the upper one fourth of memory) then processing block 130 is executed. Processing block 130 maps the address to a physical address in channel 0. Since the two MSBs are both "1"s, the address is not interleaved, but is mapped into a physical address in the upper half of channel 0 memory.

When the MSBs of the program address are not both "1"s (indicating that the address will reside in the lower three fourths of memory) then processing block 140 is executed. Processing block 140 performs a summing modulo 3 operation on bits 31:7 of the program address as described in detail above. The remainder resulting from this operation is the channel number the program address will mapped into.

Processing continues with processing block 150 which maps the program address into a physical address within the channel identified by processing block 140. The process 100 then ends.

Referring now to Figures 3A and 3B the process for determining a channel number from a program address is shown. The process 200 begins and processing block 210 is executed.

Processing block 210 uses bits 30:7 of the program address. Bits 6:0 of the program address are not used in the determination of the channel since they refer to the byte within the memory block. Processing block 220 separates the address bits 30:7 into pairs. The pairs will be used in performing a recoding of the bits. Processing block 230 performs the 2 bit to 2 bit recoding of all the pairs.

"11" pairs are recoded to a "00", while all other pair combinations remain the same.

Processing block 240 separates the recoded pairs into groups of four. These groups of four will be used in performing a 4 bit to 2 bit reduction. Processing block 250 performs the 4 bit to 2 bit reduction in accordance with Table 5.

Decision block 260 determines whether there are 6 bits left in the address after performing the 4 bit to 2 bit reductions. When there are more then 6 bits left, then processing block 250 is executed again. When there are 6 bits left, processing block 270 is executed.

When the result of processing block 250 is a 6 bit value, then processing block 270 is executed. Processing block 270 sets the interim channel number to zero. Processing block 280 pre-pends the interim channel number to the 6 bit answer from processing block 250 resulting in an 8 bit value. Processing block 290 performs 4 bit to 2 bit reduction on this 8 bit value to produce a 4 bit result.

Processing block 300 performs 4 bit to 2 bit reduction on the 4 bit result from processing block 290. This operation results in a 2 bit result.

Decision block 310 determines whether the result from processing block 300 was "00". If the result is not "00" then processing continues at processing block 320. When the result from processing block 300 is "00", then the channel number is set to the current value of the interim channel number. The channel number is the memory channel the program address will be mapped into.

When the result of execution of processing block 300 is not a value of "00", then execution continues with processing block 320. In processing block 320 current value of the interim channel number is incremented. The first time through the loop comprising processing blocks 280–320 the channel number was set to zero by processing block 270. If this did not result in a zero value after the processing of block 300, then the interim channel number is incremented to a "01". If the loop of blocks 280-320 is executed again, the interim channel number would increment from a "01" to a "10".

Once processing blocks 280 through 310 produce a value of "00", the value of the interim channel number indicates the memory channel the program address will be mapped into. The process then ends.

Referring now to Figure 4, a method for performing the mapping of the program address to a physical address within one of the memory channels is shown. The process starts and decision block 410 is executed. Decision block 410 determines whether the program address references a location in the lower three/fourths of memory. If the program address references the lower three/fourths of memory then the program address will be interleaved between the three channels and processing continues at processing block 420. When the program address does not reference the lower three/fourths of memory, the program address will not be interleaved between the three channels and processing continues at processing block 420.

In processing block 420 a summing modulo 3 arithmetic operation is performed on the program address bit 31:7 to obtain the appropriate memory

channel number (0, 1 or 2). Once the memory channel has been determined, the mapping of the address into the channel will be performed.

The memory mapping operation starts in processing block 430 where a determination is made regarding the number of consecutive address bits that are all "1"s, beginning with address bit 7 and working up to address bit 31. Once this value is determined, processing continues at processing block 440.

In processing block 440, address bits 31:7 of the program address are right shifted by a predetermined amount. This amount is dependent upon the number of consecutive "1"s determined in processing block 430. A reference table (e.g., Table 3 discussed above) may be used for determining the number of right shift operations to be performed on address bits 31:7 of the program address. Alternately, the number of right shift operations may be determined by other appropriate means.

Processing continues with processing block 450 wherein a predetermined offset value is added to the right shifted address to obtain an interim physical address. The predetermined offset value may also be included in the reference table, or may be determined by other means as appropriate.

In processing block 460 bits 6:0 of the program address are appended to the interim physical address to obtain the physical address. This is the physical address within the determined memory channel that the program address references.

In processing block 470 the physical address within the selected memory channel is accessed. Processing then ends.

Processing block 480 is executed when the program address does not reference the lower three/fourths of memory. In processing block 480 the program address is mapped into a physical address in memory channel 0. This is accomplished by executing the code segment described above. In effect, the code segment takes the program address and subtracts from the program address a number which is half the amount of total memory. The result from this is the physical address within memory channel 0 that this program address maps to.

In processing block 490 the physical address within memory channel 0 is accessed. Processing then ends.

Another aspect of utilizing non-uniformly distributed DRAM configurations involves detecting in-range memory address matches. To facilitate software development and debugging, for example when a memory location is being corrupted such as by an unintentional write to a location, address range checking can be performed. When software in an ME, core processor or other NP component accesses a logical address that overlaps with any one of a software-specified logical address range, the NP DRAM controller will report a match and take appropriate action.

As discussed above, NP DRAM channels are used for packet data buffering. In one embodiment each channel is striped at 128-byte boundaries. Accesses to memory can span 128 bytes starting at any 8-byte boundary. As a result, data for a single access can be returned from two neighboring channels.

In one particular embodiment a pair of matching address registers (an upper address matching register and a lower address matching register) and a control register are used to match any access that is served by this active channel. While a pair of address matching registers is described in this embodiment, it should be understood that any number of address matching registers could be used. The upper address matching register contains the upper address value for memory range checking within the channel. The lower address matching register contains the lower address value for memory range checking within the channel. The control register is used to turn address range matching on or off, and stores data relating to the type of operation to be performed when an address range match occurs.

One or two of the 3 channels, which enqueue the DRAM requests and own the DRAM physical location for the access range, may report a match for an access. Referring now to Figure 5 a diagram 500 showing examples regarding which channel(s) will report matching is shown. Memory is interleaved such that consecutive 128 byte blocks of memory are allocated to consecutive channels. Thus a first memory block 510 is allocated to channel 0, the next 128 byte block 512 is allocated to channel 1, the next 128 byte block 514 is allocated to channel 2, the next 128 byte block 516 is allocated to channel 0 etc.

In the diagram 500 of Figure 5, a first lower and upper range, designated Lower/Upper 0 lies within the 128 byte block 510. Five different accesses are shown, all of which result in an address range match occurring. There are three rules for determining whether an address range match will occur. Given a lower

range address and an upper range address, and a starting address and an ending address for a memory access, the rules are as follows:

An address range match will occur under any of the following conditions:

a. The starting address of the memory access lies between the lower range address and the upper range address;

b. The ending address of the memory access lies between the lower range address and the upper range address; or

c. The lower range address lies between the starting address of the memory access and the ending address of the memory access.

In Access 0, since both the starting address of the memory access and the ending address of the memory access lie within the lower range address and the upper range address, channel 0 will report a range match.

Access 1 has a starting address that lies outside of the lower range address and the upper range address, but has an ending address that lies within the lower range address and the upper range address, therefore channel 0 will report a range match.

Access 2 has a starting address that lies within the lower range address and the upper range address, and an ending address that lies outside of the lower range address and the upper range address, therefore channel 0 will report a range match.

Access 3 has a starting address that lies within the lower range address and the upper range address, but has an ending address that lies outside of the lower range address and the upper range address and which extends beyond a 128 byte

boundary (e.g. from block 510 and into block 512). In this instance channel 0 will report a range match.

Access 4 has a starting address that lies before the lower range address and has an ending address that lies beyond the upper range address, thus, Access 4 extends beyond and includes the lower range address and the upper range address. Accordingly, channel 0 will report a range match.

A second lower and upper range, designated Lower/Upper 1 begins in block 510 and extends into block 512, thus traversing a 128 byte block boundary.

Access 5 has a starting address that lies within the lower range address and upper address range and has an ending address that lies beyond the upper range address. In this case, since the access matches an address range extending from in channel 0 and into channel 1, both channel 0 and channel 1 will report a range match.

Access 6 has a starting address that lies before the lower range address and has an ending address that lies beyond the upper range address, thus, Access 6 extends beyond and includes the lower range address and the upper range address. Accordingly, since the access matches an address range extending from in channel 0 and into channel 1, both channel 0 and channel 1 will report a range match.

Once an address match is identified, the system can perform one of several different actions. The core processor can be interrupted, the pending memory operation can be aborted, a halt can be executed, or an exception can be sent to

the ME. Other operations could also be performed in response to an address match occurrence. While the address match operation has been described with respect to performing software debugging, the address match mechanism could also be provided as a security measure to prevent attempts to overwrite program code by a nefarious user (i.e., a hacker).

Software is responsible for specifying appropriate logical address ranges in channels of their physical DRAM location. In a configuration where three channels are populated, 6 pairs of ranges in total can be specified in the system, i.e., 2 pairs in each channel. Alternatively, identical two ranges can be duplicated in all three channels, so that software does not have to calculate the physical locations of logical address ranges to be matched.

Referring now to Figure 6, a flow diagram of a method 600 for performing address range checking is shown. The process 600 starts and processing block 610 is executed. In processing block 610 the upper and lower addresses for the range checking are designated. The range specified by the upper and lower addresses may span a 128 byte block of memory, and therefore cross a boundary from one channel and into another channel.

Processing continues with processing block 620, where memory accesses are monitored. The beginning address of the memory access as well as the ending address of the memory access are considered as part of the monitor operation.

In decision block 630 a determination is made regarding whether the memory access falls into the range defined by the upper and lower range addresses. When the memory access is not within the range defined the upper and

lower range addresses, processing continues at block 620. When the memory access is within the range specified by the upper and lower addresses, then processing continues with processing block 640.

In processing block 640, once an address range match has occurred, an action is taken. Potential actions taken include, but are not limited to, interrupting the core processor, aborting the pending memory operation, executing a halt, or sending an exception to the ME. Other actions could also be taken in response to an address match occurring. Following processing block 640, processing ends.

Having described particular embodiments, it will now become apparent to those of ordinary skill in the art that other embodiments incorporating these concepts may be used. Additionally, the software may be embodied in a computer program product that includes a computer useable medium. For example, such a computer usable medium can include a readable memory device, such as a hard drive device, a CD-ROM, a DVD-ROM, or a computer diskette, having computer readable program code segments stored thereon. The computer readable medium can also include a communications link, either optical, wired, or wireless, having program code segments carried thereon as digital or analog signals. Accordingly, it is submitted that that the present application should not be limited to the described embodiments but rather should be limited only by the spirit and scope of the appended claims.